**Singly/Single Linked List**

**What is a linked list?**

A data structure that contains a **head**, **tail** and **length** property.

Linked Lists consist of **nodes**, and each node has a **value** and a **pointer** to another node or null.

**Comparisons with Arrays**

**Lists**

* Do not have indexes!
* Connected via nodes with a next pointer
* Random access is not allowed

**Arrays**

* Indexed in order!
* Insertion and deletion can be expensive
* Can quickly be accessed at a specific index

**Start Code to Linked List: (Starter Code)**

**Node:**

* Linked list is just a collections of "nodes"
* Each node just stores piece of data --> "value"
* Each node stores reference to next Node --> "next"
* Each node is an Object consisting 2 keys, *value: ‘data’ & next: reference to nxt node (Node)*
* A Linked list has a pointer to the head of the list, a pointer to the tail. Those a just a variable and then a length.

**Code:**

class Node {

constructor(val) {

this.val = val;

this.next = null;

}

}

let first = new Node("Hi");

first.next = new Node("There");

first.next.next = new Node("This");

first.next.next.next = new Node("is");

first.next.next.next.next = new Node("Chandan");

**Output:**Node {val: 'Hi', next: Node} //Type **first** in Console. First executing above code.

**Pushing:-**

Adding a new node to the end of the Linked List!

**Pushing Approach:**

* Create A Node i.e. Object consisting keys --> value: 'data' & next: null i.e. reference to the next node
* If(there is No Node i.e. head = null)

-- Assign head = 1st Node

-- Assign tail = 1st Node

-- Increase Length (Imagine: Creating Forward Arrow)

-- Return List

* If(There is Node i.e head pointing to --> First Node)  
  -- Since, tail is Still assigning to the Previous Node consisting --> value: 'data' & next: null

-- So, assigning 2nd Node to the next key of Previous Node --> tail.next = 2nd Node

To point tail to the 2nd node, Assign 2nd Node to the tail.

-- Increase Length (Imagine: Creating Forward Arrow)  
-- Return List

**Pushing Pseudocode**

* This function should accept a value
* Create a new node using that value that we passed in to the function.  
  (So if we pass in 50, you should make a new node where the value of that node is 50.)
* If there is no head property on the list, set the head and tail to be the newly created node
* Otherwise set the **next property** on the tail to be the new node and set/update the tail property on the list to be the newly created node
* Increment the length by one
* Return the linked list

**Pushing Code**

class Node {

constructor(val) {

this.val = val;

this.next = null;

}

}

class SinglyLinkedList {

constructor() {

this.head = null;

this.tail = null;

this.length = 0;

}

push(val){

let newNode = new Node(val);

if(!this.head){

this.head = newNode;

this.tail = this.head;

}else{

this.tail.next = newNode;

this.tail = newNode;

}

this.length++;

return this;

}

}

let list = new SinglyLinkedList();  
list.push("Hi");  
list.push("There");  
list.push("This");  
list.push("is");  
list.push("Chandan");

**Output:**SinglyLinkedList {head: Node, tail: Node, length: 5}

**Popping:-**

Removing a node from the end of the Linked List!

**Popping Approach:**

* First of all, POP Operation (Removing Elmnt from End) can only be perform if their is at least one Node in the list.
* So, lets assume there's already a created list having 5 nodes i.e. length=5, head pointing to 1st Node & tail pointing to last Node.
* Each node storing value & reference to the next node.
* For Popping:
* Create a current & newTail pointer
* Initially, Set current & newTail pointing to the 1st node like head is pointing.
* Now Traverse through the list till the end of the Node. (Moving current pointer till the end i.e. where old tail is pointing to the last node, having next.node ===null)

Set newTail pointer to the current node

Set current pointer to the next node.

This will continuously moving, current pointer first to the next node and behind newTail pointer pointing just back to the current node.

* Stop the Loop when current pointer reached to the last node i.e. current pointer points node as old tail pointer pointing to the last node, having reference node === null
* Since, newTail pointer still pointing to the node just Previous to the current node(last node). We'll set Old tail pointer back to the node poinying by newtail pointer.
* Now, Node of old tail pointer, having reference node (i.e. current node), set it to null. Linke, this.tail.next = null. To breakout the current Node from the List.
* Decrease the Length of the list.
* Return current node. This node would be the pop out node from the list.

**Popping Pseudocode**

* If there are no nodes in the list, return undefined
* Loop through the list until you reach the tail
* Set the next property of the 2nd to last node to be null
* Set the tail to be the 2nd to last node
* Decrement the length of the list by 1
* Return the value of the node removed

**Popping Code**

class Node {

constructor(val) {

this.val = val;

this.next = null;

}

}

class SinglyLinkedList {

constructor() {

this.head = null;

this.tail = null;

this.length = 0;

}

push(val){

let newNode = new Node(val);

if(!this.head){

this.head = newNode;

this.tail = this.head;

}else{

this.tail.next = newNode;

this.tail = newNode;

}

this.length++;

return this;

}

pop(){

if(!this.head) return undefined;

let current = this.head,

newTail = current;

while(current.next){

newTail = current;

current = current.next;

}

this.tail = newTail;

this.tail.next = null;

this.length--;

//When All Nodes POP Out.

if(this.length ===0){

this.head = null;

this.tail = null;

}

return current;

}

}

let list = new SinglyLinkedList();

list.push("Hi");  
list.push("There");  
list.push("This");  
list.push("is");  
list.push("Chandan");

list.pop(); ***//This will pop out each node per execution***

**Output:**Node {val: 'Chandan', next: null}

**Shifting:-**

Removing a new node from the beginning of the Linked List!

**Shifting Approach:**

* First of all, Shift Operation (Remove Elmnt from beginning) can only be perform if their is at least one Node in the list.
* So, lets assume there's already a created list having 5 nodes i.e. length=5, head pointing to 1st Node & tail pointing to last Node.

Each node storing value & reference to the next node.

* For Shifting:

Create a current & newHead pointer

Initially, Set current & newHead poiting to the 1st node like head is pointing.

* Now, Set newHead pointer to the reference of Head Node. i.e. newHead = this.head.next.
* Set/move, head pointer to the node pointing by the newHead pointer.
* Since, current pointer still pointing to the node which was Previously pointing by the head pointer.
* Decrease the Length of the list.
* Return current node. This node would be the shifted out node from the list.

**Shifting Pseudocode**

* If there are no nodes, return undefined
* Store the current head property in a variable
* Set the head property to be the current head's next property
* Decrement the length by 1
* Return the value of the node removed

**Shifting Code**

class Node {

constructor(val) {

this.val = val;

this.next = null;

}

}

class SinglyLinkedList {

constructor() {

this.head = null;

this.tail = null;

this.length = 0;

}

push(val){

let newNode = new Node(val);

if(!this.head){

this.head = newNode;

this.tail = this.head;

}else{

this.tail.next = newNode;

this.tail = newNode;

}

this.length++;

return this;

}

shift(){

if(!this.head) return undefined;

let current = this.head,

newHead = current;

newHead = this.head.next;

this.head = newHead;

this.length--;

//When All Nodes POP Out.

if(this.length ===0){

this.head = null;

this.tail = null;

}

return current;

}

}

let list = new SinglyLinkedList();

list.push("Hi");

list.push("There");

list.push("This");

list.push("is");

list.push("Chandan");

list.shift(); ***//This will shift out each node per execution***

**Output:**Node {val: 'Hi', next: Node}

**Unshifting:-**

Adding a new node to the beginning of the Linked List!

**Unshifting Approach**

* Create A Node i.e. Object consisting keys --> value: 'data' & next: null i.e. reference to the next node
* If(there is No Node i.e. this.head == null or this.length==null)

Assign head = 1st Node

Assign tail = 1st Node

Increase Length (Imagine: Creating Forward Arrow)

Return List

* If(There is Node i.e head pointing to --> First Node)

Create a newHead pointer pointing to the newly created Node;

Since, Old head pointer still pointing to the first node.

Now, Set old first node as the reference of the newNode pointing by newHead pointer. i.e. newHead.next = this.head;

Set, Old head pointer to the node pointing by the newHead pointer;

* Increase Length
* return this (this pointing to the class SinglyLinkedList as a Current context)

**Unshifting Pseudocode**

* This function should accept a value
* Create a new node using the value passed to the function
* If there is no head property on the list, set the head and tail to be the newly created node
* Otherwise set the newly created node's next property to be the current head property on the list
* Set the head property on the list to be that newly created node
* Increment the length of the list by 1
* Return the linked list

**Unshifting Code**

class Node {

constructor(val) {

this.val = val;

this.next = null;

}

}

class SinglyLinkedList {

constructor() {

this.head = null;

this.tail = null;

this.length = 0;

}

unshift(val){

let newNode = new Node(val);

if(!this.head){

this.head = newNode;

this.tail = this.head;

}

else{

let newhead = newNode;

newhead.next = this.head;

this.head = newhead;

}

this.length++;

return this;

}

}

let list = new SinglyLinkedList();

list.unshift("Hi");  
list.unshift("There");  
list.unshift("This");  
list.unshift("is");  
list.unshift("Chandan");

**Output:**SinglyLinkedList {head: Node, tail: Node, length: 5}

**Get:-**

Retrieving a node by it's position in the Linked List!

**Get Approach:**

* For getting any node at any position, we've to start traversing the list from 1st node.
* Accept Index Number/Position by Getting the node from the list
* index num shouldn't be negative Or Greater than or equal (Initialize list this.length = 0) to the length of the list.
* Initialize counter with 0.
* Create a current pointer for traversnig over the list till counter !== index.

Set the current pointer to the 1st node as head pointer pointing to it.

Now to move the current pointer, keep setting up current pointer to the reference node stored by current node pointing by current pointer. i.e. current = current.next;

And increment the counter for each iteration

* Loop will stop as counter === index
* return current (this would return the required node as per the accepted position).

**Get Pseudocode**

* This function should accept an index
* If the index is less than zero or greater than or equal to the length of the list, return null
* Loop through the list until you reach the index and return the node at that specific index

**Get Code:**

class Node {

constructor(val) {

this.val = val;

this.next = null;

}

}

class SinglyLinkedList {

constructor() {

this.head = null;

this.tail = null;

this.length = 0;

}

push(val){

let newNode = new Node(val);

if(!this.head){

this.head = newNode;

this.tail = this.head;

}else{

this.tail.next = newNode;

this.tail = newNode;

}

this.length++;

return this;

}

get(index){

if(index < 0 || index >=this.length) return null;

let counter = 0,

current = this.head;

while(counter!==index){

current = current.next;

counter++;

}

return current;

}

}

let list = new SinglyLinkedList();

list.push("Hi");  
list.push("There");  
list.push("This");  
list.push("is");  
list.push("Chandan");

**Output:**Node {val: 'Chandan', next: null}

**Set:-**

Changing the value of a node based on it's position in the Linked List

**Set Pseudocode**

* This function should accept a value and an index
* Use your get function to find the specific node.
* If the node is not found, return false
* If the node is found, set the value of that node to be the value passed to the function and return true

**Set Approach:**

* SET Operations change/set the value of the available node in the list. Accept(index & val);
* If(index is negative or >= length of the list) return null/undefined.
* Create a current pointer assing it to the head node and Create & Initialize counter with 0
* Increase counter by 1 as per move of current pointer to the nodes using loop till counter !== index;
* As Loop terminate,

SET the value of the current node with the new val.

* Return the list. (Check the value of node at the required has changed or not ).

**Get Code:**

class Node {

constructor(val) {

this.val = val;

this.next = null;

}

}

class SinglyLinkedList {

constructor() {

this.head = null;

this.tail = null;

this.length = 0;

}

push(val){

let newNode = new Node(val);

if(!this.head){

this.head = newNode;

this.tail = this.head;

}else{

this.tail.next = newNode;

this.tail = newNode;

}

this.length++;

return this;

}

set(index, val){

if(index<0 || index>=this.length) return null;

let counter=0,

current = this.head;

while(counter !== index){

current = current.next;

counter ++;

}

current.val = val;

return this;

}

}

let list = new SinglyLinkedList();

list.push("Hi");  
list.push("There");  
list.push("This");  
list.push("is");  
list.push("Chandan");

list.set(0, "Hello"); ***//set() method execute set operation on list.***

**Output:**SinglyLinkedList {head: Node, tail: Node, length: 5} **//Changed “hi” to “hello”**

**Insert:-**

Adding a node to the Linked List at a specific position

**Insert Approach:-**

* Insert Operations Insert a new node at any index/position (Index may be, 0, last, or at any position accross the list )
* If(index is negative or > length of the list) return null/undefined.
* If(index is equal to 0 ) execute the unshift() method.
* Otherwise, If(index is equal to the length of the list ) execute the push() method.
* Otherwise, Create a current pointer & a prevCurrent pointer(Move behind current pinter) assin both to the head node and Create & Initialize counter with 0
* Increase counter by 1 as per move of current pointer to the nodes using loop till counter !== index;
* As Loop terminate,

Create a newNode

Assign reference of the newNode to the node pointing by the current pointer.

Move back the current pointer to the newNode;

Assign reference of the Node pointing by the prevCurrent pointer to the current Node pointing by the current pointer;

* Return current (Inserted Node).

**Insert Pseudocode**

* If the index is less than zero or greater than the length, return false
* If the index is the same as the length, push a new node to the end of the list
* If the index is 0, unshift a new node to the start of the list
* Otherwise, using the get method, access the node at the index - 1
* Set the next property on that node to be the new node
* Set the next property on the new node to be the previous next
* Increment the length
* Return true

**Insert Code:**

class Node {

constructor(val) {

this.val = val;

this.next = null;

}

}

class SinglyLinkedList {

constructor() {

this.head = null;

this.tail = null;

this.length = 0;

}

push(val){

let newNode = new Node(val);

if(!this.head){

this.head = newNode;

this.tail = this.head;

}else{

this.tail.next = newNode;

this.tail = newNode;

}

this.length++;

return this;

}

unshift(val){

let newNode = new Node(val);

if(!this.head){

this.head = newNode;

this.tail = this.head;

}

else{

let newHead = newNode;

newHead.next = this.head;

this.head = newHead;

}

this.length++;

return this;

}

insert(index, val){

if(index < 0 || index >this.length) return null;

else if(index===0) this.unshift(val);

else if(index === this.length) this.push(val);

else{

let counter = 0,

current = this.head,

prevCurrent = current;

while(counter!==index){

prevCurrent = current;

current = current.next;

counter++;

}

let newNode = new Node(val);

newNode.next = current;

current = newNode;

prevCurrent.next= current;

this.length++;

return current;

}

}

}

let list = new SinglyLinkedList();

list.push("Hi");  
list.push("There");  
list.push("This");  
list.push("is");  
list.push("Chandan");

list.insert(6, "last"); ***//Execute insert method .***  
list.insert(4, "Mid");  
list.insert(6, "last");

**Output:**SinglyLinkedList {head: Node, tail: Node, length: 8}

**Remove:-**

Removing a node from the Linked List at a specific position

**Remove Approach**

* Remove Operations Remove a new node at any index/position (Index may be, 0, last, or at any position accross the list )
* If(index is negative or > length of the list) return null/undefined.
* If(index is equal to 0 ) execute the shift() method.
* Otherwise, If(index is equal to the length of the list ) execute the pop() method.
* Otherwise, Create a current pointer & a prevCurrent pointer(Move behind current pinter) assin both to the head node and Create & Initialize counter with 0
* Increase counter by 1 as per move of current pointer to the nodes using loop till counter !== index;
* As Loop terminate,

Assign reference of the Node pointing by the prevCurrent pointer to the reference of the node pointing by the current pointer;

* Decrease the length--;
* Return current (Removed Node).

**Remove Pseudocode**

* If the index is less than zero or greater than the length, return undefined
* If the index is the same as the length-1, pop
* If the index is 0, shift
* Otherwise, using the get method, access the node at the index - 1
* Set the next property on that node to be the next of the next node
* Decrement the length
* Return the value of the node removed

**Remove Code:**

class Node {

constructor(val) {

this.val = val;

this.next = null;

}

}

class SinglyLinkedList {

constructor() {

this.head = null;

this.tail = null;

this.length = 0;

}

push(val){

let newNode = new Node(val);

if(!this.head){

this.head = newNode;

this.tail = this.head;

}else{

this.tail.next = newNode;

this.tail = newNode;

}

this.length++;

return this;

}

pop(){

if(!this.head) return null;

let current = this.head,

newTail = current;

while(current.next!==null){

newTail = current;

current = current.next;

}

this.tail = newTail;

this.length--;

if(this.length===0){

this.head = null;

this.tail = null;

}

return current;

}

shift(){

if(!this.head) return null;

let current = this.head;

this.head = current.next;

if(this.length===0){

this.head = null;

this.tail = null;

}

this.length--;

return current;

}

remove(index){

if(index < 0 || index >=this.length){ return null;}

else if(index===0) { this.shift(); return true;}

else if(index === this.length-1) { this.pop(); return lastNode;}

else{

let counter = 0,

current = this.head,

prevCurrent = current;

while(counter!==index){

prevCurrent = current;

current = current.next;

counter++;

}

prevCurrent.next= current.next;

this.length--;

return current;

}

}

}

let list = new SinglyLinkedList();

list.push("Hi");  
list.push("There");  
list.push("This");  
list.push("is");  
list.push("Chandan");

list.remove(0); ***//Execute remove method***  
list.remove(4);  
list.remove(2);

**Output:**Node {val: 'Hi', next: Node}  
Node {val: 'Chandan', next: null}  
Node {val: 'is', next: Node}

**Reverse:-**

Reversing the Linked List in place!

**Information:**

* Reverse Operation of a List:
  + Just swap the head pointer with the tail pointer. (Although, in real, node of head & tail pointer/head Node & tail Node get swaped.)
  + Then, Change the Direction of reference of next node storing by each node in the list. As per the General property of head Node & tail Node.
* General property of head node & tail Node:
  + In tail Node, reference of next node always be null whereas reference of head node might be null (In case of 1 Node) or might be any node.
  + reference of Direction of next node always start from head and end to tail.

**Approach:**

* Swap the head Pointer with tail pointer. (In practical, we actually swap the head node & tail node with head & tail pointer).

Ex:  
 head tail

Before swapping: Hi, There, This, is, Chandan

tail head

After swapping: Hi, There, This, is, Chandan

* + But the General property of head Node & tail Node i.e. (In tail Node, reference of next node always null whereas reference of head node might be null or any node).
* Create 3 pointers, prev(previous Pointer), nodePtr(Node Pointer) & next (Next Pointer).
  + Initialization: prev = null, nodePtr already with tail pointer & next not Initialized
* Start loop to traverse through the list till the length of the list:
  + Set the next pointer to the reference node of node pointer.
  + Change the reference of node pointer with the node pointing by the previous pointer.
  + Set the previous Pointer to the node pointer.
  + Set the node pointer to the next pointer.
* Steps:
  + Steps has shown in the code file in commented form.

**Reverse Pseudocode**

* Swap the head and tail
* Create a variable called next
* Create a variable called prev
* Create a variable called node and initialize it to the head property
* Loop through the list
* Set next to be the next property on whatever node is
* Set the next property on the node to be whatever prev is
* Set prev to be the value of the node variable
* Set the node variable to be the value of the next variable
* Once you have finished looping, return the list

**Reverse Code:**

class Node {

constructor(val) {

this.val = val;

this.next = null;

}

}

class SinglyLinkedList {

constructor() {

this.head = null;

this.tail = null;

this.length = 0;

}

push(val){

let newNode = new Node(val);

if(!this.head){

this.head = newNode;

this.tail = this.head;

}else{

this.tail.next = newNode;

this.tail = newNode;

}

this.length++;

return this;

}

reverse(){

let node = this.head;

this.head = this.tail;

this.tail = node;

let next,

prev = null;

for (let i = 0; i < this.length; i++) {

next = node.next;

node.next = prev;

prev = node;

node = next;

}

return this;

}

print(){

let arr = [],

current = this.head;

while(current){

arr.push(current.val);

current = current.next;

}

return arr;

}

}

let list = new SinglyLinkedList();

list.push("Hi");

list.push("There");

list.push("This");

list.push("is");

list.push("Chandan");

list.reverse(); ***//To reverse list***  
list.print(); ***//To print the reversed list***

**Output:**

*SinglyLinkedList {head: Node, tail: Node, length: 5}*

**head**: Node {val: 'Chandan', next: Node}  
**length**: 5  
**tail**: Node {val: 'Hi', next: null}

['Chandan', 'is', 'This', 'There', 'Hi']

**Big O of Singly Linked Lists**

Insertion - **O(1)**

Removal - It depends.... **O(1)** or **O(N)**

Searching - **O(N)**

Access - **O(N)**

**RECAP**

* Singly Linked Lists are an excellent alternative to arrays when insertion and deletion at the beginning are frequently required
* Arrays contain a built in index whereas Linked Lists do not
* The idea of a list data structure that consists of nodes is the foundation for other data structures like Stacks and Queues